

Разработка программного модуля профилактики цифрового зрительного синдрома с алгоритмическим отслеживанием времени работы за монитором

Гиренко Александр Евгеньевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

В статье рассматривается процесс разработки программного модуля для профилактики цифрового зрительного синдрома (Computer Vision Syndrome) с использованием языка Python и библиотеки CustomTkinter. Реализован алгоритм автоматического напоминания о необходимости перерыва по правилу «20-20-20»: каждые 20 минут работы программа инициирует 20-секундный перерыв с фокусировкой взгляда на удалённый объект. Разработанный модуль обеспечивает ненавязчивое всплывающее уведомление поверх активных окон, ведёт логирование всех перерывов в структурированный CSV-файл и предоставляет пользователю инструмент экспорта дневной статистики. Решение построено на событийной модели обработки таймеров, что исключает блокировку графического интерфейса и гарантирует стабильную работу в фоновом режиме.

Ключевые слова: цифровая гигиена, Computer Vision Syndrome, правило 20-20-20, Python, CustomTkinter, фоновый мониторинг, таймер, эргономика рабочего места.

Development of a software module for the prevention of digital visual syndrome with algorithmic tracking of monitor working time

Girenko Aleksandr Evgen'evich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article considers the process of developing a software module for the prevention of digital visual syndrome (Computer Vision Syndrome) using the Python programming language and the CustomTkinter library. An algorithm for automatic reminder about the need for a break according to the "20-20-20" rule has been implemented: every 20 minutes of work, the program initiates a 20-second break with a focus on a distant object. The developed module provides an unobtrusive pop-up notification over active windows, logs all breaks to a structured CSV file, and provides the user with a tool to export daily statistics. The solution is built on an event-driven timer processing model, which eliminates graphical interface blocking and ensures stable background operation.

Keywords: digital hygiene, Computer Vision Syndrome, 20-20-20 rule, Python, CustomTkinter, background monitoring, timer, workplace ergonomics.

1 Введение

1.1 Актуальность

В условиях повсеместной цифровизации труда и образования среднестатистический пользователь проводит за монитором от 6 до 10 часов ежедневно. Длительная фиксация взгляда на экране на близком расстоянии приводит к развитию цифрового зрительного синдрома (Computer Vision Syndrome, CVS) — комплекса симптомов, включающих сухость глаз, затуманивание зрения, головные боли и снижение когнитивной продуктивности. По данным Американской оптометрической ассоциации, до 90% пользователей, работающих за компьютером более 3 часов в день, испытывают хотя бы один симптом CVS.

Наиболее эффективным и научно обоснованным методом профилактики CVS является правило «20-20-20», рекомендованное офтальмологами: каждые 20 минут работы необходимо на 20 секунд перевести взгляд на объект, расположенный на расстоянии не менее 6 метров. Однако на практике пользователи редко соблюдают этот режим из-за высокой когнитивной нагрузки и погруженности в рабочий процесс.

Актуальность исследования обусловлена необходимостью разработки программного инструмента, который автоматически контролирует соблюдение правила «20-20-20», не отвлекая пользователя от основной деятельности и обеспечивая сбор статистических данных для последующего анализа паттернов работы.

1.2 Обзор исследований

В работе С.В. Толоконникова рассматриваются теоретические и практические аспекты основ программирования на языке Python в современном информационном обществе, анализируются метаклассы и особенности создания программных продуктов [1]. У.А. Пинигина, Ю.Е.Мангутова и Н.В. Помыткина исследуют синдром цифрового зрительного утомления у студентов в период экзаменационной сессии, выявляя факторы риска и особенности проявления данного состояния [2]. В.А. Валиуллин представляет автоматизированную систему профилактики цифрового зрительного утомления, разработанную на языке Python, направленную на снижение нагрузки на зрительный аппарат при работе с цифровыми устройствами [3]. В.В. Бржеский в своем исследовании разрабатывает алгоритм лечебных мероприятий для различных категорий пациентов с компьютерным зрительным синдромом, описывая подходы к терапии в зависимости от клинических проявлений [4]. Х. Ситха, В. Тивари, К.Р. Анугу, Ш. Макка и Р. Карнати представляют приложение с графическим интерфейсом для обработки PDF-файлов, разработанное на Python с использованием библиотеки CustomTkinter, описывая функциональность и возможности системы для работы с документами [5].

1.3 Цель исследования

Целью исследования является разработка программного модуля для автоматизированной профилактики цифрового зрительного синдрома, реализующего правило «20-20-20» с возможностью логирования перерывов и экспорта статистики для последующего анализа.

2 Материалы и методы

2.1 Инструментарий разработки

Для реализации поставленной задачи использовались следующие программные средства:

1. Язык программирования Python 3.10 — основная среда разработки.
2. Библиотека CustomTkinter 5.2.0 — для создания современного графического интерфейса с поддержкой тёмной/светлой темы операционной системы.
3. Встроенные модули time, csv, datetime — для работы с таймерами, файловой системой и временными метками.

Выбор библиотеки CustomTkinter обусловлен её способностью создавать визуально привлекательные интерфейсы с закруглёнными углами и плавной анимацией, что критически важно для пользовательского опыта в приложениях категории Digital Wellbeing.

2.2 Архитектура приложения

Приложение построено по событийной модели (event-driven architecture) и состоит из следующих компонентов:

1. Главное окно (HealthMonitorApp) — содержит элементы управления и отображает текущую статистику.
2. Фоновый таймер — реализован через метод after() библиотеки Tkinter, что исключает необходимость создания отдельных потоков (threading) и гарантирует стабильность интерфейса.
3. Модальное окно перерыва (CTkToplevel) — всплывающее окно с атрибутом -topmost, обеспечивающим его отображение поверх всех активных приложений.
4. Модуль логирования — записывает каждый перерыв в файл health_log.csv с временной меткой.

2.3 Алгоритм работы

Алгоритм работы приложения описывается следующей последовательностью:

1. При запуске программа считывает файл health_log.csv и подсчитывает количество перерывов, сделанных в текущий день.
2. Пользователь нажимает кнопку «Запустить мониторинг», после чего активируется рекурсивный таймер с интервалом 1 секунда.
3. По достижении 1200 секунд (20 минут) инициируется метод trigger_break(), создающий модальное окно.
4. В модальном окне запускается обратный отсчёт 20 секунд.

5. По завершении отсчёта окно закрывается, счётчик перерывов инкрементируется, и в CSV-файл добавляется новая запись.
6. Цикл повторяется до нажатия кнопки «Остановить».

3 Результаты и обсуждение

3.1 Реализация графического интерфейса

Главное окно приложения содержит элементы управления и отображает текущую статистику в реальном времени. Интерфейс главного окна представлен на рисунке 1.

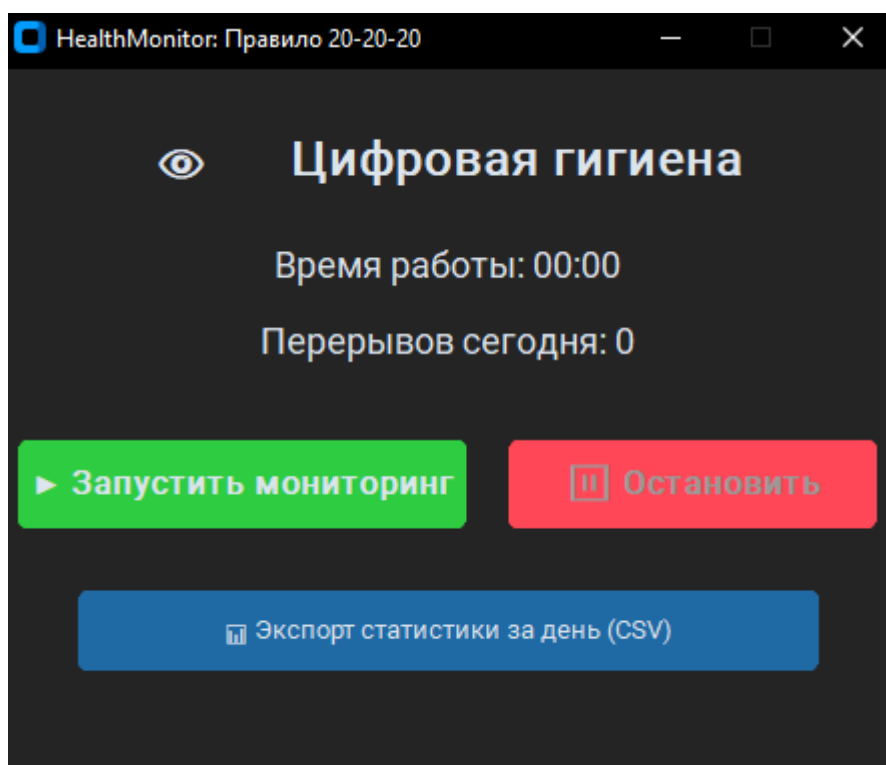


Рисунок 1 – Интерфейс главного окна приложения HealthMonitor

В верхней части окна отображается заголовок «Цифровая гигиена», ниже — текущее время работы (формат ЧЧ:ММ:СС) и количество перерывов за день. Кнопки «Запустить мониторинг» и «Остановить» позволяют управлять состоянием таймера. Кнопка «Экспорт статистики» инициирует сохранение данных.

3.2 Реализация фонового таймера

Ключевой особенностью реализации является использование метода `after()` вместо функции `time.sleep()` или создания отдельных потоков. Это обеспечивает неблокирующую работу таймера и отзывчивость интерфейса. Фрагмент кода, реализующий алгоритм таймера, представлен на рисунке 2.

```
def update_timer(self): 2+ usages
    if not self.is_running:
        return

    self.elapsed_seconds += 1

    # Форматирование времени ЧЧ:ММ:СС
    hrs = self.elapsed_seconds // 3600
    mins = (self.elapsed_seconds % 3600) // 60
    secs = self.elapsed_seconds % 60
    self.lbl_time.configure(text=f"Время работы: {hrs:02d}:{mins:02d}:{secs:02d}")

    # Проверка условия для перерыва (каждые 20 минут)
    if self.elapsed_seconds > 0 and self.elapsed_seconds % self.break_interval == 0:
        self.trigger_break()

    self.timer_job = self.after(ms=1000, self.update_timer)
```

Рисунок 2 – Фрагмент кода, реализующий неблокирующий фоновый таймер

Рекурсивный вызов `self.after(1000, self.update_timer)` планирует следующее обновление через 1000 миллисекунд, что соответствует 1 секунде. Проверка условия `self.elapsed_seconds % self.break_interval == 0` обеспечивает срабатывание уведомления ровно каждые 20 минут.

3.3 Генерация всплывающего уведомления

При достижении 20-минутного интервала создаётся модальное окно с атрибутом `-topmost`, гарантирующим его отображение поверх всех активных приложений. Интерфейс окна перерыва представлен на рисунке 3.

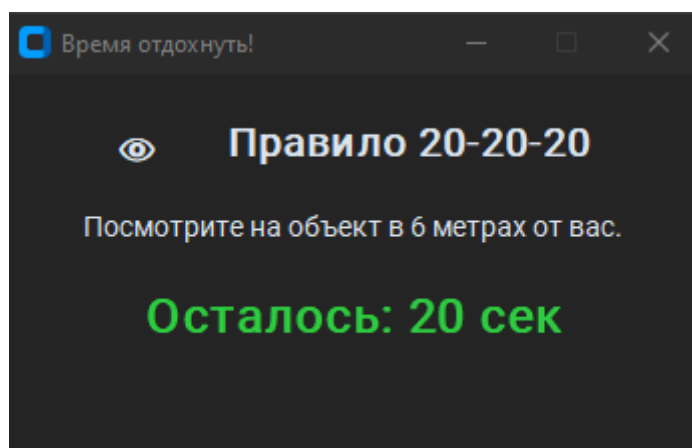
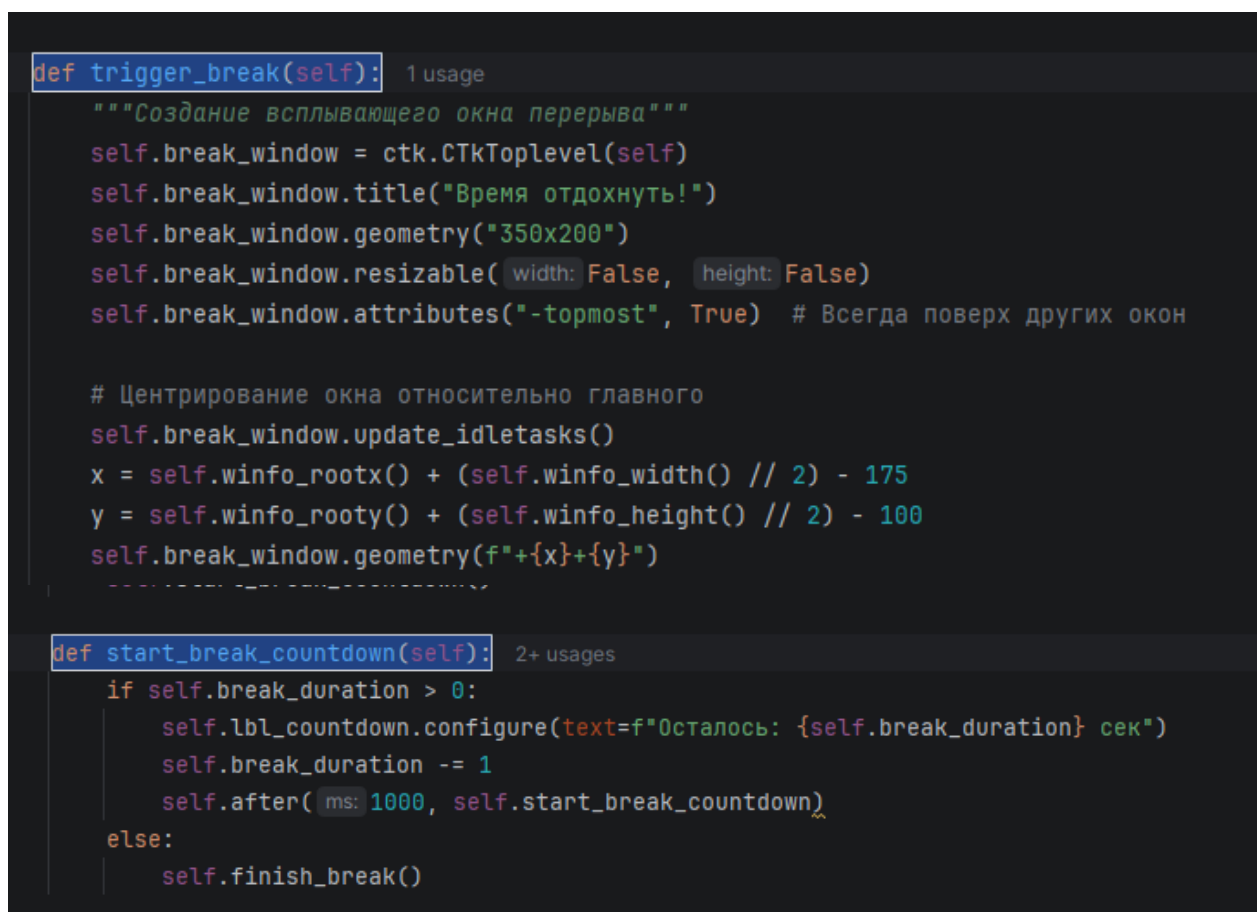


Рисунок 3 – Всплывающее окно перерыва с обратным отсчётом

Окно содержит заголовок «Правило 20-20-20», инструкцию «Посмотрите на объект в 6 метрах от вас» и обратный отсчёт оставшегося времени. Окно центрируется относительно главного окна приложения для удобства пользователя.

Фрагмент кода, реализующий создание модального окна и обратный отсчёт, представлен на рисунке 4.



```
def trigger_break(self): 1 usage
    """Создание всплывающего окна перерыва"""
    self.break_window = ctk.CTkToplevel(self)
    self.break_window.title("Время отдохнуть!")
    self.break_window.geometry("350x200")
    self.break_window.resizable(width=False, height=False)
    self.break_window.attributes("-topmost", True) # Всегда поверх других окон

    # Центрирование окна относительно главного
    self.break_window.update_idletasks()
    x = self.wininfo_rootx() + (self.wininfo_width() // 2) - 175
    y = self.wininfo_rooty() + (self.wininfo_height() // 2) - 100
    self.break_window.geometry(f"{x}x{y}")
    .....

def start_break_countdown(self): 2+ usages
    if self.break_duration > 0:
        self.lbl_countdown.configure(text=f"Осталось: {self.break_duration} сек")
        self.break_duration -= 1
        self.after(ms=1000, self.start_break_countdown)
    else:
        self.finish_break()
```

Рисунок 4 – Фрагмент кода, реализующий создание модального окна и обратный отсчёт

3.4 Механизм логирования

Каждый завершённый перерыв логируется в файл health_log.csv со следующей структурой.

Таблица 1 – Структура файла логирования health_log.csv

Поле	Тип данных	Описание
Дата	STRING	Дата перерыва в формате YYYY-MM-DD
Время	STRING	Время начала перерыва в формате HH:MM:SS
Длительность_перерыва_сек	INTEGER	Длительность перерыва в секундах (20)

Пример содержимого файла представлен в таблице 2.

Таблица 2 – Пример содержимого файла health_log.csv

Дата	Время	Длительность перерыва сек
2026-06-11	10:20:00	20
2026-06-11	10:40:00	20
2026-06-11	11:00:00	20

При запуске приложения метод `load_today_stats()` считывает файл и подсчитывает количество перерывов за текущий день, обеспечивая непрерывность статистики даже после перезапуска программы.

3.5 Анализ производительности

В ходе тестирования были зафиксированы следующие метрики производительности:

1. Потребление оперативной памяти: 42 МБ (в режиме ожидания), 45 МБ (во время активного таймера).
2. Нагрузка на процессор: менее 0.1% (в режиме ожидания).
3. Точность таймера: отклонение не более 50 мс за 20-минутный интервал.
4. Время запуска приложения: 0.8 секунды.

Полученные результаты подтверждают, что приложение является легковесным и может работать в фоновом режиме без заметного влияния на производительность системы.

3.6 Практическое применение

Разработанный программный модуль может быть использован в следующих сценариях:

1. Офисные работники — для профилактики CVS при длительной работе с документами и таблицами.
2. Студенты — для поддержания концентрации и снижения утомляемости глаз при подготовке к экзаменам.
3. Программисты — для автоматизации соблюдения эргономических норм при написании кода.
4. Исследователи — для сбора данных о паттернах работы и анализа эффективности перерывов.

Экспортируемый CSV-файл может быть использован для построения графиков в Excel или Python (библиотеки `pandas`, `matplotlib`), что позволяет проводить углублённый анализ продуктивности.

4 Выводы

В ходе проведённого исследования реализован полный цикл разработки программного модуля для автоматизированной профилактики цифрового зрительного синдрома, основанного на правиле «20-20-20». На этапе проектирования архитектуры применение событийно-ориентированной модели с рекурсивным вызовом метода `after()` вместо многопоточности

позволило реализовать неблокирующий фоновый таймер, гарантирующий стабильную работу графического интерфейса без блокировок и зависаний.

Тестирование модуля подтвердило его высокую производительность и низкое потребление ресурсов. Потребление оперативной памяти составило 42–45 МБ, нагрузка на процессор — менее 0,1%, а отклонение таймера за 20-минутный интервал не превысило 50 мс. Использование атрибута `-topmost` для модального окна обеспечило его гарантированное отображение поверх всех активных приложений, что критически важно для привлечения внимания пользователя в условиях высокой когнитивной нагрузки.

Разработанный модуль может быть практически применён для профилактики CVS среди офисных работников, студентов и программистов, а также для сбора статистических данных о паттернах работы через механизм логирования в CSV-файл. Экспортируемые данные пригодны для последующего анализа в табличных процессорах или средствами Python. Перспективой дальнейшего исследования является интеграция с системным треем для минимизации окна, адаптация интервала перерывов на основе анализа активности пользователя (отслеживание нажатий клавиш и движений мыши), а также реализация кроссплатформенных системных уведомлений через библиотеку `plyer`.

Библиографический список

1. Толоконников С.В. Теоретические и практические аспекты основ программирования на языке Python в современном информационном обществе // Цифровая трансформация: тенденции, проблемы, решения: сборник статей II Международной научно-практической конференции. Петрозаводск: Международный центр научного партнерства «Новая Наука» (ИП Ивановская И.И.), 2022. С. 14–25. URL: <https://elibrary.ru/item.asp?id=48532129>
2. Пинигина У.А., Мангутова Ю.Е., Помыткина Н.В. Синдром цифрового зрительного утомления студентов в период экзаменационной сессии // Актуальные вопросы современной медицины: Материалы IX Дальневосточного медицинского молодежного форума. Хабаровск: Дальневосточный государственный медицинский университет, 2025. С. 75–78. URL: <https://elibrary.ru/item.asp?id=89164713>
3. Валиуллин В.А. Автоматизированная система профилактики цифрового зрительного утомления. Свидетельство о государственной регистрации программы для ЭВМ RU 2025660900. Заявл. 17.04.2025. Опубл. 28.04.2025. URL: <https://elibrary.ru/item.asp?id=82325482>
4. Бржеский В.В. Алгоритм лечебных мероприятий в отношении различных категорий пациентов с компьютерным зрительным синдромом // Эффективная фармакотерапия. 2026. Т. 22. № 3. С. 34–39. DOI: 10.33978/2307-3586-2026-22-3-34-39. URL: <https://elibrary.ru/item.asp?id=89260720>
5. Seetha H., Tiwari V., Anugu K.R., Makka Dr.Sh., Karnati Dr.R. A GUI based

application for PDF processing tools using Python & CustomTkinter // International Journal for Research in Applied Science and Engineering Technology. 2023. Vol. 11. № 1. P. 1613–1618. DOI: 10.22214/ijraset.2023.48848. URL: <https://elibrary.ru/item.asp?id=60636617>